Московский авиационный институт
(Национальный исследовательский университет)

Институт: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа № 4

Тема: Основы метапрограммирования

Студент: Пивницкий Даниэль Сергеевич
Группа: 80-206
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи

Разработать шаблоны классов согласно варианту задания. (6-угольник, 8-угольник,Треугольник)
Параметром шаблона должен являться скалярный тип данных задающий тип данных для оси координат.
Классы должны иметь только публичные поля. В классах не должно быть методов, только поля.
Фигуры являются фигурами вращения (равнобедренными), за исключением трапеции и прямоугольника.
Для хранения координат фигур необходимо использовать шаблон std::pair.

2. Описание программы

Изначально создается базовый класс Figure задающий общий принцип структуры для классов — наследников(треугольника, 6-угольника и 8-угольника).

Наследование позволяет избежать дублирования лишнего кода при написании классов, т. к. класс может использовать переменные и методы другого класса как свои собственные.

В данном случае класс Figure является абстрактным — он определяет интерфейс для переопределения методов другими классами.

3. Набор тестов

test_01

1
1 1
1 1
1 1
2
1 1
1 1
1 2
1 1
1 1
1 1
3
1 1
1 2
1 2
1 1

1 1
1 1
1 2
1 1
4
3
2 2
1 0
0 0
q

test_02

4
6
1 1
2 2
3 3
4 4
5 5
6 6
q

4. Результаты выполнения тестов

choose option (m for menu, q to quit): 1

1 1

1 1

1 1

(1 1) (1 1) (1 1)

0

(1 1)

choose option (m for menu, q to quit): 2

1 1

1 1

1 2

1 1

1 1

1 1

(1 1) (1 1) (1 2) (1 1) (1 1) (1 1)

0

(1 1.16667)

choose option (m for menu, q to quit): 3

1 1

1 2

1 2

1 1

1 1

1 1

1 2

1 1

(1 1) (1 2) (1 2) (1 1) (1 1) (1 1) (1 2) (1 1)

0

(1 1.375)

choose option (m for menu, q to quit): 4

enter number of angles: 3

2 2

1 0

0 0

(2 2) (1 0) (0 0)

1

(1 0.666667)

choose option (m for menu, q to quit): q

C:\Users\Daniel\source\repos\Lab41\Debug\Lab41.exe (процесс 9716) завершил работу с кодом 0.

Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрыть консоль при остановке отладки".

Нажмите любую клавишу, чтобы закрыть это окно…

choose option (m for menu, q to quit): 4

enter number of angles: 6

1 1

2 2

3 3

4 4

5 5

6 6

(1 1) (2 2) (3 3) (4 4) (5 5) (6 6)

0

(3.5 3.5)

choose option (m for menu, q to quit): q

C:\Users\Daniel\source\repos\Lab41\Debug\Lab41.exe (процесс 9620) завершил работу с кодом 0.

Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрыть консоль при остановке отладки".

Нажмите любую клавишу, чтобы закрыть это окно…

5. Листинг программы

```
//

//  main.cpp

//  lab4

//  Variant 15

//  M8o-206Б-19

//  Created by Daniel Pivnitskiy on 10.10.2020.

//  github.com/SLAST1

//  Copyright © 2020 Daniel Pivnitskiy. All rights reserved.

//

/*

Разработать шаблоны классов согласно варианту задания.

Параметром шаблона должен являться скалярный тип данных задающий тип данных для оси координат.

Классы должны иметь только публичные поля. В классах не должно быть методов, только поля.

Фигуры являются фигурами вращения (равнобедренными), за исключением трапеции и прямоугольника.

Для хранения координат фигур необходимо использовать шаблон  std::pair.

*/

#include <iostream>

#include "templates.h"

#include "point.h"
```

```
#include "figures.h"
#include <vector>

int main() {
	char option = '0';
	while (option != 'q') {
		std::cout << "choose option (m for menu, q to quit): ";
		std::cin >> option;
		switch (option) {
		default:
			std::cout << "no such option, try m for menu" << std::endl;
			break;
		case 'q':
			break;
		case 'm': {
			std::cout << "1) triangle" << '\n'
				<< "2) hexagon" << '\n'
				<< "3) octagon" << '\n'
				<< "4) tuple" << '\n'
				<< "5) dop" << std::endl;
			break;
		}
		case '1': {
			figures<Triangle<double>>(std::cin, std::cout);
			break;
```

```
                }
                case '2': {
                        figures<Hexagon<double>>(std::cin, std::cout);
                        break;
                }
                case '3': {
                        figures<Octagon<double>>(std::cin, std::cout);
                        break;
                }
                case '4': {
                        figures<std::tuple<point<double>>>(std::cin, std::cout);
                        break;
                }
                case '5': {
                        std::cout << is_ok<std::string, char*>::value << std::endl;
                        std::cout << is_ok<float, int>::value << std::endl;
                        std::cout << is_ok<char, std::vector < double>>::value << std::endl;
                }
                }
        }
        return 0;
}
//
//  point.h
//  lab4
```

```
//  Variant 15
//  М8о-206Б-19
//  Created by Daniel Pivnitskiy on 10.10.2020.
//  github.com/SLAST1
//  Copyright © 2020 Daniel Pivnitskiy. All rights reserved.
//
#ifndef OOP_EXERCISE_04_POINT_H
#define OOP_EXERCISE_04_POINT_H

#include <iostream>
#include <type_traits>
#include <cmath>



template<class T>
struct point {
	T x;
	T y;
	point<T>& operator=(point<T> A);
};

template<class T>
std::istream& operator>>(std::istream& is, point<T>& p) {
	is >> p.x >> p.y;
	return is;
```

```
}

template<class T>
std::ostream& operator<<(std::ostream& os, point<T> p) {
	os << '(' << p.x << ' ' << p.y << ')';
	return os;
}

template<class T>
point<T> operator+(const point<T>& A, const point<T>& B) {
	point<T> res;
	res.x = A.x + B.x;
	res.y = A.y + B.y;
	return res;
}

template<class T>
point<T>& point<T>::operator=(const point<T> A) {
	this->x = A.x;
	this->y = A.y;
	return *this;
}

template<class T>
point<T> operator+=(point<T>& A, const point<T>& B) {
```

```
	A.x += B.x;
	A.y += B.y;
	return A;
}

template<class T>
point<T> operator/=(point<T>& A, const double B) {
	A.x /= B;
	A.y /= B;
	return A;
}

template<class T>
double point_length(point<T>& A, point<T>& B) {
	double res = sqrt(pow(B.x - A.x, 2) + pow(B.y - A.y, 2));
	return res;
}

template<class T>
struct is_point : std::false_type {};

template<class T>
struct is_point<point<T>> : std::true_type {};
#endif
```

```
//
//  classes.h
//  lab4
//  Variant 15
//  M8o-206Б-19
//  Created by Daniel Pivnitskiy on 10.10.2020.
//  github.com/SLAST1
//  Copyright © 2020 Daniel Pivnitskiy. All rights reserved.
//
#ifndef OOP_EXERCISE_04_CLASSES_H
#define OOP_EXERCISE_04_CLASSES_H

#include "point.h"
#include <type_traits>
#include <iostream>

template <class T>
class Triangle {
public:
	point<T> dots[3];
	int size = 3;
	explicit Triangle<T>(std::istream& is) {
		for (auto& dot : dots) {
			is >> dot;
		}
```

```
        }
};

template <class T>
class Hexagon {
public:
        point<T> dots[6];
        int size = 6;
        explicit Hexagon<T>(std::istream& is) {
                for (auto& dot : dots) {
                        is >> dot;
                }
        }
};

template <class T>
class Octagon {
public:
        point<T> dots[8];
        int size = 8;
        explicit Octagon<T>(std::istream& is) {
                for (auto& dot : dots) {
                        is >> dot;
                }
        }
```

```
};
#endif

//
//  templates.h
//  lab4
//  Variant 15
//  M8o-206Б-19
//  Created by Daniel Pivnitskiy on 10.10.2020.
//  github.com/SLAST1
//  Copyright © 2020 Daniel Pivnitskiy. All rights reserved.
//
#ifndef OOP_LAB4_FIGURES_H
#define OOP_LAB4_FIGURES_H

#include <tuple>
#include <type_traits>
#include <cassert>

#include "point.h"
#include "classes.h"

template<class T, class = void>
struct has_dots : std::false_type {};
```

```
template<class T>
struct has_dots<T, std::void_t<decltype(std::declval<const T&>().dots)>> : std::true_type {};

template<class T>
struct is_figurelike_tuple : std::false_type {};

template<class Head, class... Tail>
struct is_figurelike_tuple<std::tuple<Head, Tail...>> :
	std::conjunction<is_point<Head>, std::is_same<Head, Tail>...> {};

template<size_t Id, class T>
void tuple_print(const T& object, std::ostream& os) {
	if constexpr (Id >= std::tuple_size<T>::value) {
	}
	else {
		os << std::get<Id>(object) << " ";
		tuple_print<Id + 1>(object, os);
	}
}
template <class T>
void printout(const T& object, std::ostream& os) {
	if constexpr (has_dots<T>::value) {
		for (auto dot : object.dots) {
			os << dot << " ";
		}
```

```
        }
        else if constexpr (is_figurelike_tuple<T>::value) {
                tuple_print<0>(object, os);
        }
        else {
                throw std::logic_error("ERROR! Perhaps tuple is incorrect");
        }
}
template<size_t Id, class T>
point<double> tuple_center(const T& object) {
        if constexpr (Id >= std::tuple_size<T>::value) {
                return point<double> {0, 0};
        }
        else {
                point<double> res = std::get<Id>(object);
                return res + tuple_center<Id + 1>(object);
        }
}
template <class T>
point<double> center(const T& object) {
        point<double> res{ 0.0, 0.0 };
        int i = 0;
        if constexpr (has_dots<T>::value) {
                for (auto dot : object.dots) {
                        res += dot;
```

```
                ++i;
            }
            res /= i;
            return res;
        }
        else if constexpr (is_figurelike_tuple<T>::value) {
            res = tuple_center<0>(object);
            res /= std::tuple_size_v<T>;
            return res;
        }
        else {
            throw std::logic_error("ERROR! Perhaps tuple is incorrect");
        }
}
template<size_t Id, class T>
double tuple_area(const T& object) {
        if constexpr (Id >= std::tuple_size<T>::value - 1) {
            return 0.0;
        }
        else {
            double res = (std::get<Id>(object).x * std::get<Id + 1>(object).y) -
                (std::get<Id + 1>(object).x * std::get<Id>(object).y);
            return res + tuple_area<Id + 1>(object);
        }
}
```

```
////

template<class U, class V, class = void>

struct is_ok : std::false_type {};


template<class U, class V>

struct is_ok <U, V,

        std::void_t<decltype(U(std::declval<const V&>())) >

    > : std::true_type {};

////

template <class T>

double area(const T& object) {

    double res = 0.0;

    if constexpr (has_dots<T>::value) {

        for (int i = 0; i < object.size - 1; ++i) {

            res += (object.dots[i].x * object.dots[i + 1].y) - (object.dots[i + 1].x * object.dots[i].y);

        }

        res += (object.dots[object.size - 1].x * object.dots[0].y) - (object.dots[0].x * object.dots[object.size - 1].y);

        return std::abs(res) / 2;

    }

    else if constexpr (is_figurelike_tuple<T>::value) {

        res = tuple_area<0>(object);

        res += (std::get<std::tuple_size<T>::value - 1>(object).x * std::get<0>(object).y) - (std::get<0>(object).x * std::get<std::tuple_size<T>::value - 1>(object).y);
```

```
            return std::abs(res) / 2;
        }
        else {
            throw std::logic_error("ERROR! Perhaps tuple is incorrect");
        }
}
#endif

//
//  figures.h
//  lab4
//  Variant 15
//  М8о-206Б-19
//  Created by Daniel Pivnitskiy on 10.10.2020.
//  github.com/SLAST1
//  Copyright © 2020 Daniel Pivnitskiy. All rights reserved.
//
#ifndef OOP_EXERCISE_04_FIGURES_H
#define OOP_EXERCISE_04_FIGURES_H

template<class T>
void figures(std::istream& is, std::ostream& os) {
    if constexpr (has_dots<T>::value) {
        T object(is);
        printout(object, os);
```

```
        os << std::endl;
        os << area(object) << std::endl;
        os << center(object) << std::endl;
}
else if constexpr (is_figurelike_tuple<T>::value) {
        size_t s;
        os << "enter number of angles: ";
        is >> s;
        switch (s) {
        case 3: {
                point<double> fig[3];
                for (auto& i : fig) {
                        is >> i;
                }
                auto [a, b, c] = fig;
                auto object = std::make_tuple(a, b, c);
                printout(object, os);
                os << std::endl;
                os << area(object) << std::endl;
                os << center(object) << std::endl;
                break;
        }
        case 6: {
                point<double> fig[6];
                for (auto& i : fig) {
```

```
                is >> i;
        }
        auto [a, b, c, d, e, f] = fig;
        auto object = std::make_tuple(a, b, c, d, e, f);
        printout(object, os);
        os << std::endl;
        os << area(object) << std::endl;
        os << center(object) << std::endl;
        break;
}
case 8: {
        point<double> fig[8];
        for (auto& i : fig) {
                is >> i;
        }
        auto [a, b, c, d, e, f, g, h] = fig;
        auto object = std::make_tuple(a, b, c, d, e, f, g, h);
        printout(object, os);
        os << std::endl;
        os << area(object) << std::endl;
        os << center(object) << std::endl;
        break;
}
default:
        throw std::logic_error("incorrect number of angles, try 3, 4 or 8");
```

```
            }
        }
}
#endif
```

6. Выводы

В данной лабораторной я получил навыки работы с шаблонами и кортежами. Во многих случаях удобно использование кортежи. Например, кортежи позволяют легко определять и работать с функциями, возвращающими одно или более значений.

7. Список литературы

1.Руководство по написанию кода на C++ [Электронный ресурс].
URL:https://metanit.com/cpp/tutorial/
Дата обращения: 10.09.2019

2.Документация по C++ [Электронный ресурс]. URL:
https://docs.microsoft.com/ru-ru/cpp
Дата обращения 12.09.2019